SCTE·ISBE CABLE-TEC EXPO 2018 ATLANTA, GA OCTOBER 22-25

2018 Fall Technical Forum SCTE·ISBE • NCTA • CABLELABS

Using AI to Improve the Customer Experience

A Virtual Assistant Chatbot

A Technical Paper prepared for SCTE•ISBE by

Bernard Burg,
Fan Liu,
Abel Villca Roque,
Sunil Srinivasa,
Ryan March,
Comcast
1050 Enterprise way, Sunnyvale CA 94089
+1 408 900 85 75
bernard_burg@comcast.com

Tianwen Chen,
Comcast
1110 Vermont Av NW, Washington DC 20005

Table of Contents

List of Figures

Introduction

AI and Machine Learning (ML) are becoming pervasive, allowing new applications for chatbots. This paves the way for operational transformations in the field of the cable telecommunication industry, where chatbots will soon be able to field some customer contacts to solve their issues in real-time -- thereby reducing waiting queues while enhancing service quality, informed by consistent answers and considering all available network information in real time.

This article describes the steps service providers can use to build such a chatbot. It and drills into the AI/ML elements in charge of performing installation diagnostics, predicting the behavior of the systems and users, and pinpointing the root causes critical to fixing issues before they even become visible to customers. These highly specialized AI/ML algorithms feed their propositions into a decision engine (DE), so as to understand a much larger context, and apply game theory to make optimal choices for the customer.

1. Chatbot Architecture

A chatbot is a computer program that provides an interface that allows customers to interact with machine learning systems via auditory or textual methods. Through Chatbots, customers can report service issues and get personalized answers or help, using the best knowledge of our machine learning systems. Further, Chatbots can obviate the need for customers to navigate websites or call for support. An overview of the Chatbot is provided in Figure 1.

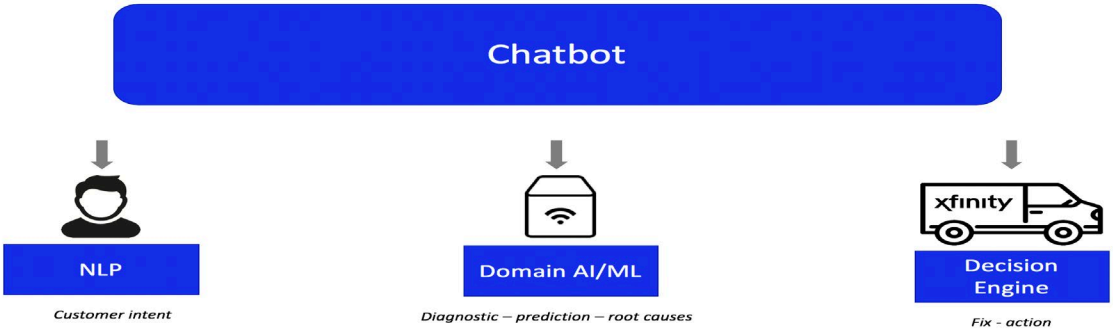


Figure 1: Overview of ChatBot

A chatbot dialog unfolds as follows: A customer interacts, either by typing a query into a mobile device, or by directly talking into a microphone which transmits the voice signal to a speech-to-text algorithm to produce a script of the query. This query is analyzed by a Natural Language Processing (NLP) module that extracts the intent of the customer. In some cases, these intents are implicit in the query and need to be translated by NLP into an explicit statement, often formulated in technical jargon. For example, a query like “it takes very long to download images” might be translated into a more generic intent like “internet speed slow,” known in the domain knowledge.

This intent is forwarded to the Machine Learning/AI domain for investigation. These ML/AI algorithms have access to a wealth of information, such as high speed internet data, in-home

WiFi data, and many kinds of system and video errors including RDK and XRE. Ideally, all of this information is stored in a data lake, with nationwide coverage and a history duration spanning over several months to years. These data lakes allow ML algorithms to learn the customer behavior or performance of our current networks, and to extract salient events, called "features," which in turn enable algorithms to assess the situation of a customer through a kind of fingerprinting.

One method used by ML to fix issues is actually to identify "error fingerprints" from the past and learn how they were fixed. All things being equal, applying the same fix should solve the same issue with high probability. Additionally, ML algorithms can predict future calls or issues that customers are experiencing based on similar "fingerprints" seen in the past. Domain AI/ML finally presents its diagnostics, predictions and recommendations of actions to the DE (decision engine.)

The DE gets recommendations from domain AI/ML as input features. These recommendations might be incoherent, in competition with or even in contradiction with each other, given that they're created by highly specialized ML algorithms -- each of which has a deep but narrow understanding of the world. The DE aims to understand a larger context to select the right decision.

In its simplest implementation, a DE can be implemented by a rules engine that can choose the best solution for a problem. However, these rules engines are static solutions, and would constantly need re-evaluations over time to make sure they still are optimal. A superior implementation of the decision engine uses a recommendation system to take into consideration larger context domains, along with domain knowledge, to continuously choose the optimal action and fix the issue reported by the customer. Our particular implementation uses what's called a "multi-arm bandit algorithm."

2. Introduction to Machine Learning

In this section, we define the machine learning algorithms used in this paper. Machine learning algorithms try to learn the underlying patterns in data. Depending on the ways that ML algorithms learn patterns, they are categorized into three families: supervised learning, unsupervised learning and reinforcement learning.

2.1. Supervised Learning

Supervised learning is a guided learning approach, meaning that it learns the pattern of the data that could optimally predict the provided labels. In our system, we use classifiers to discern similarities and differences in data, and to assign them to categories based on similarity. Examples of such classification algorithms include identifying objects in a video frame, identifying the underlying sentiment in a customer service message, or associating a log message from a set-top with a specific error class. The technologies powering classifiers range from the simple -- decision trees and random forest algorithms-- to the very complex, such as deep neural networks. The choice amongst available technologies is typically related to the level of complexity and the number of features in the underlying data. Image classification, for instance,

has been shown to benefit greatly by neural networks and its derivative technologies, such as convolutional neural nets.

2.2. Unsupervised Learning

In contrast to supervised learning, unsupervised learning aims to determine underlying patterns without any sort of guidance or provided labels. In our system, we use clustering algorithms to group similar data into clusters, and the clusters do not have any preset labels. They are typically used to understand the behavior of data or to look for any significant deviations in data. For example, clustering algorithms could be used to look at anonymized smart home data to build user profiles, such as early risers, late risers etcetera, based on common behaviors. Clustering algorithms range from the simple, such as k-means clustering, to the complex, like agglomerative hierarchical clustering.

2.3. Reinforcement Learning

Reinforcement learning is a type of ML inspired by behavioral psychology, and concerned with how software agents take actions in an environment so as to maximize some notion of cumulative reward. Reinforcement learning is studied in many disciplines, including control theory, game theory, simulation-based optimization, and more.

Reinforcement Learning is also directly applicable to the operations improvement in the cable telecommunication industry, as our networks collectively qualify as complex systems to be optimized, despite the impossibility of building a mathematical model of such a system. Early implementations of reinforcement learning have been studied in optimal control theory to tackle this very issue.

Reinforcement learning can also be used in gaming theory to select the best action to perform while optimizing gains. For cable operators, this translates into selecting the best actions to perform on an account, while minimizing the disruptions of service. Such an approach is used in our Decision Engine.

3. Domain AI

Three domain-specific AI modules are under development:

- High Speed Data
- WiFi
- Video domain models including XRE, which stands for Cross-platform Runtime Environment, and is a platform-independent protocol for distributed applications.

This paper describes the example of XRE. Three problems are assessed in this study. First, how to react after an XRE error or a sequence of XRE appears on a device. Second, how can consequences of such XRE errors or sequences be predicted, in terms of how important are they to our customers, and how much they affect our service? Third, how can these errors be prevented from happening in the future, and can problems be solved before they even surface?

3.1. XRE React: What to do after an XRE Error?

This first study investigates three actions that can be performed after observation of an XRE error or a sequence of XRE errors for a single device during a time window of an hour. An example of an XRE sequence for user a is a_1= {XRE-03059, XRE-03059, XRE-10007}. Whereas at the same time, user b might receive the sequence b_1 = {XRE-10007}, and user c might not receive any error during the time window, hence $c_1 = \varnothing$. All of these sequences represent fingerprints of the users' statuses, and they can be used to perform predictions.

Three independent ML models are used to predict the effectiveness of three actions that may be performed to fix the XRE errors (see Figure 2):

- **System Refresh**: account refresh + cable card refresh + firmware reset + reboot
- **System Restart**: simple box reboot
- **Do Nothing**: estimates the XRE error attrition over time, and recommends not to act

The goal of each of these actions is to eradicate the XRE errors. Success is claimed if there are no errors in a duration of one hour after the action has been performed.

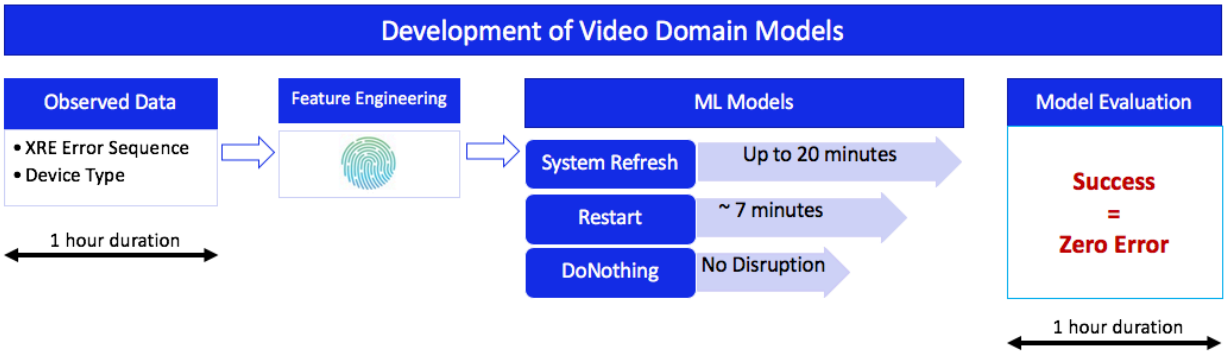


Figure 2: System Refresh, Restart and DoNothing models

For each of these actions, a supervised ML algorithm learns a classifier to organize samples into two classes: Success or failure. During the prediction phase, new examples that have never been seen are presented to the classifier, which will predict the probability of success of the action (see Figure 3).

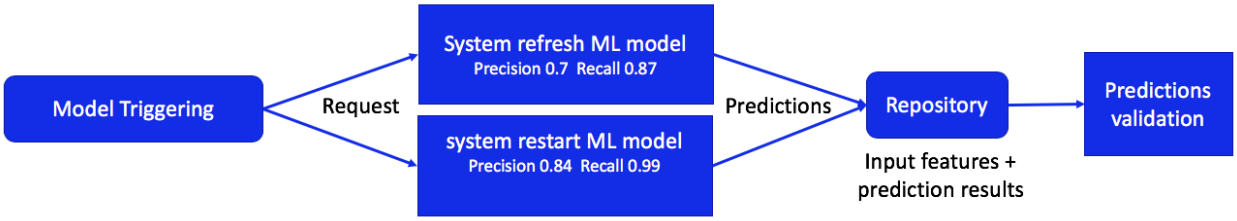


Figure 3: System Refresh and Restart Deployment Models

Current ML results allow the predicted performance of the system refresh model with 70% of precision and 87% of recall, meaning that 70% of the recommended system refreshes

successfully eradicate the XRE errors, and 87% of the effective system refreshes are captured by the ML model. See a visual representation of precision and recall in Figure 4: Precision, Recall Similarly, the ML results predict the performance of a system restart model with a precision of 84% and a recall of 99%.

The outputs of these models are stored in a repository for assessing their validities when the actions, system refresh or restart, are performed. These actions can either be performed by the customers, or by reinforcement learning models executing the ML recommendation directly. The reinforcement learning models can check the validities of the actions immediately for self-calibration. In essence, reinforcement learning is comparable to a feedback loop in control systems.

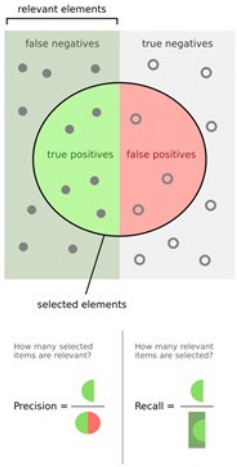


Figure 4: Precision, Recall

3.2. XRE Predict: Who needs our help now?

Comcast measures customer satisfaction by using the Net Promoter Score (NPS). NPS scores send a very clear message regarding the number of interactions required to fix an issue. Customers are fairly happy when systems work flawlessly or get repaired before having to call. First call resolutions are generally well accepted, but mark a dip in NPS. Given the impact of repeat interactions on NPS scores, it makes much sense to study the severity of the XRE fingerprints in aiming to predict and later preempt the customer calls.

3.2.1. Call Predictions

The call prediction scenario for ML is similar to the previous example, as it also uses supervised learning. In this case, ML learns to predict the calls based on XRE errors that occurred over 24 hours. The modeling of these errors is also more precise, as it records each of the errors into hourly buckets. In Figure 5, the studied device received {11 XRE-00021, 5 XRE-03056, 33 XRE-03059} at 15:00, {7 XRE-00021, 5 XRE-03056, 22 XRE-03059} at 16:00, etc. The goal of the ML is to differentiate the callers from the non-callers based on these XRE models, and, given a new sequence the ML algorithm never encountered before, predict if this customer will call.

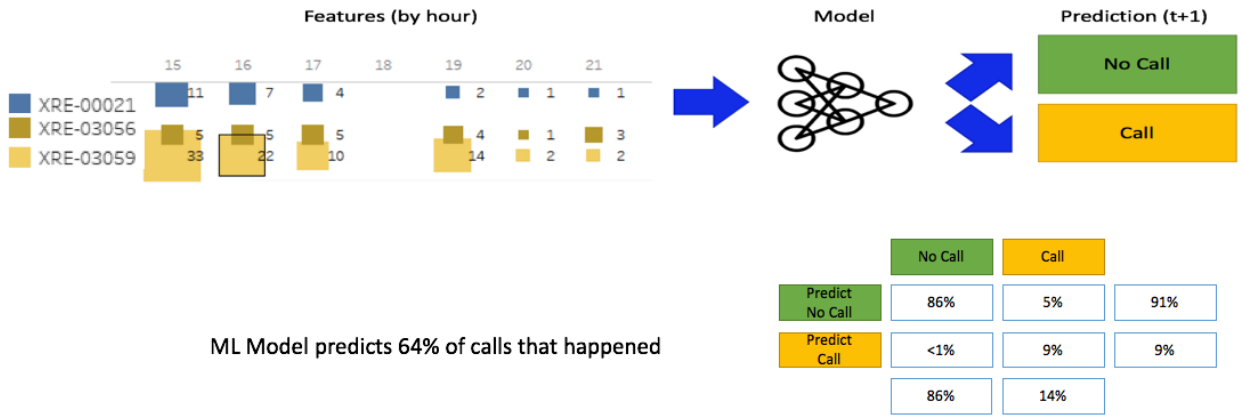


Figure 5: Call Predictions Model Flow and Results

The ML model predicted 64% of the calls that will happen, which is quite a valuable result because the learning sets are unbalanced -- meaning there are very few callers as compared to non-callers. But the most exploitable result of this study is the high accuracy in predicting a call. If the ML predicts a call, it is wrong in less than 1% of the cases. However, the algorithm only catches nine percent of the total number of calls. This result is interesting in the sense that we can predict some calls with a very high accuracy, and thus are poised to preempt these calls by fixing the underlying issues.

3.2.2. Silent Sufferers

Previous methods used for detecting quality of service issues are typically based on user feedback, gathered through chats and calls. Implicitly this means that no news means all is right, right? Wrong.

There exists a small category of customers who never use our communication channels, even while facing issues. Hence, they are called silent sufferers. These customers are undetectable with previous methods, they are unaccounted for in NPS, and user histories indicate that silent sufferers often unsubscribe without ever calling.

No supervised learning method can address silent sufferers. We present instead the use of an unsupervised ML method. The design principle is to introduce a distance measurement between users. The ML measures a fingerprint of a single XRE error over a 24 hour period of time. The unsupervised method used is the k-means algorithm, clustering users into classes of similarity according to this fingerprint measurement. Experiments show that 90% of the users are clustered into the main class (see Figure 6). The devices of these users show a nominal behavior where everything works fine, up to specification. Other classes are called outliers, which group similar users. Some of these classes prove to have a nominal behavior whereas others might face unusual behaviors, for example, degraded service. These classes are called the sufferer classes.

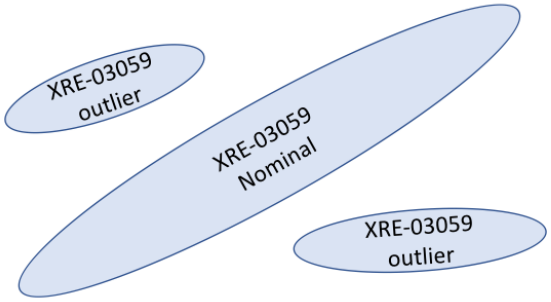


Figure 6: Initial Clusters

We defined several types of logic to combine the classes of sufferers across different XRE errors into meta-classes, so as to reinforce the predictive power of the weak predictor classes. Such results are described in Figure 7. Since the k-means algorithm performs clustering regardless of calls, the resulting meta-classes of sufferers contain, in general, both callers and silent sufferers. For each of these classes independently, an algorithm studies the causes mentioned by the callers to gather a better understanding of this class. The knowledge acquired through the callers is shared to the silent sufferers and allows action on their issues without having them to ever call.

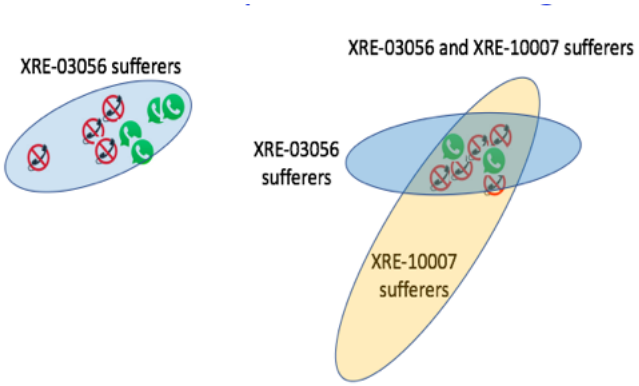


Figure 7: Meta Clusters with callers and silent sufferers

3.3. XRE Prevent: Can we solve the problems before they surface?

The best way to prevent XRE errors is to understand their root causes and to tackle them before they can propagate and morph into customer-impacting XRE errors. It can be done with supervised learning but requires advanced feature engineering, because such models rely on a number of data sources of different natures/semantics. In particular see Figure 8:

- Hardware data: XRE telemetry, RDK telemetry, xFi data transfer telemetry
- Networking data: Spectra data in the cable, WOPR, PHT data
- Application data: XRE messages
- Behavioral data: Anonymized click and tune data generated by users.

In a first approach, these data were bucketed hourly, in the same manner as in 3.2.1. Because of the large number and the heterogeneity of the data sources, this approach is unlikely to work because of the different sampling rates in each of these data sources. Some of the data are sampled hourly while others are sampled at the micro-second level (e.g. the clicks and tunes). By using an hour as the common denominator, the millisecond-rate information of the clicks and tunes is washed away, and the clicks and tunes lose most of their precious information.

To overcome such difficulties, we developed our own asynchronous sampling methods into our models, inspired by [3]. Asynchronous sampling allows the capture of all the information without using buckets.

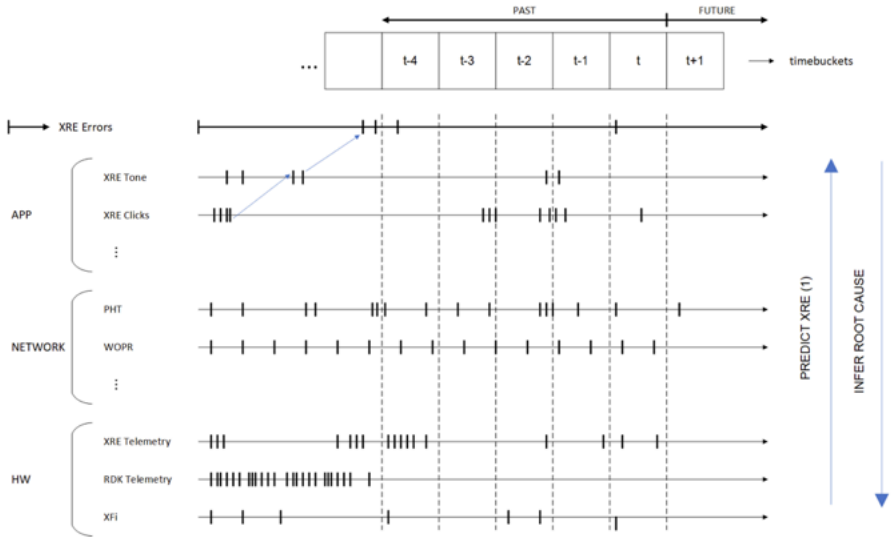


Figure 8: Root Cause Analysis and Asynchronous Sampling

4. Decision Engine (DE)

Each of the `depth-first' domain AI algorithms provides recommendations based on their narrow but deep understanding of the building blocks of a cable operator's infrastructure. The decision engine operates in a `breadth-first' fashion: It is in charge of collecting these recommendations, their contexts, the user's intent and making sense of it all, to execute the best actions that will improve the customer experience.

4.1. Overview

Figure 9 gives an overview of the decision engine workflow. The customer intent is acquired through the NLP module; the customer context is acquired through a virtual assistant guiding users in the resolution of their tasks; the customer's experience is provided by the domain AI algorithms presented earlier -- they analyze many sources of raw data, perform a diagnostic of the situation and propose actions to fix issues. Together, the customer's intent, context and experience form the overall context or state of the environment. Given this state, the job of the decision engine is to take the right action at the minimal cost. In the process, it might have to make trade-offs between the actions proposed by the domain AI, as some of these actions, though successful, might come at a cost in terms of user impact (see details in Figure 2). For

example, the restart of a set top box takes about 7 minutes, whereas the more powerful refresh operation fixes potentially more issues but can take (worst case) up to 20 minutes. In some cases, the domain AI even calculates the natural error attrition of the system and suggests not to perform any action, so as to not adversely impact the user.

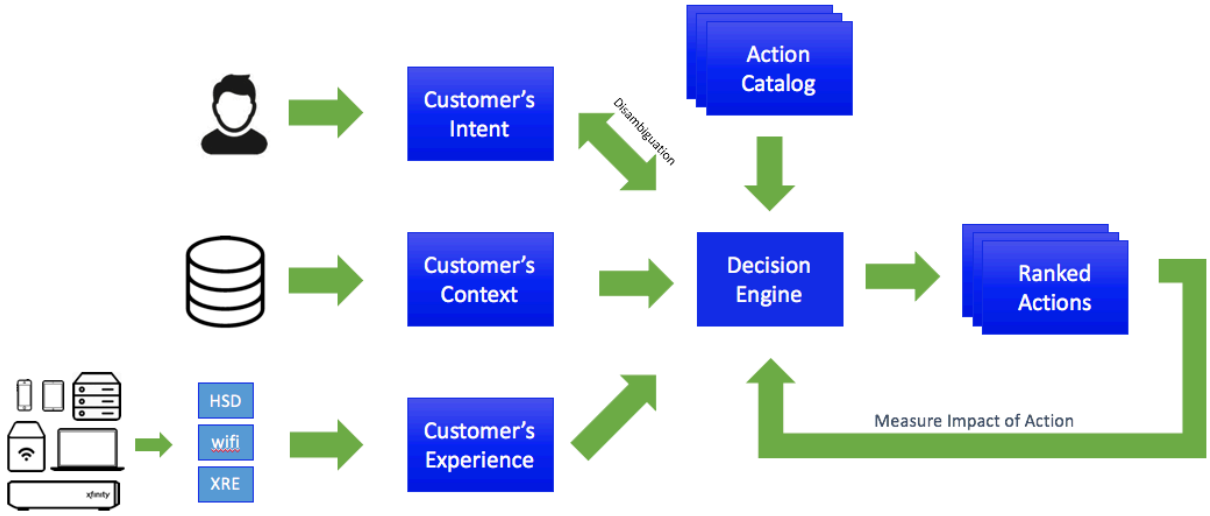
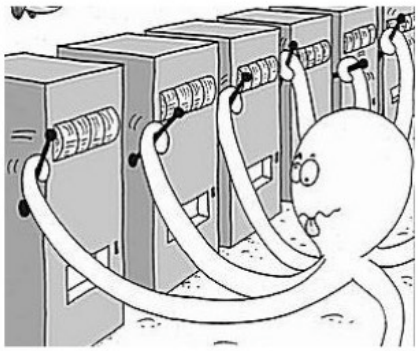


Figure 9: Decision Engine Data Flow

For a given context, the decision engine consults its action catalog, assesses the benefits and costs of the actions, and finds the best tradeoff to fulfill the customer's intents. In reinforcement learning terminology, the decision engine takes as input a state (or context), and predicts the best action(s) to take so as to minimize cost (or maximize reward).

4.2. Multi-Armed Bandit (MAB) Algorithm Overview

The operation of the decision engine is driven by the multi-armed bandit (MAB) algorithm [1], which is widely used for single-step decision-making problems. The name `multi-armed bandit' references a gambler (generally, a bandit) at a casino with several arms, trying to play the right slot machines so as to optimize their winnings.


source: Microsoft Research

MAB is often depicted in cartoons as an octopus in front of several slot machines (each assumed to provide a specific payout) and trying to use its several arms to determine the slot machine that provides the highest payout.

An intrinsic characteristic of MAB algorithms is the **`exploration/exploitation'** tradeoff. In the context of the slot machine payout, this is what the terms mean:

- Exploitation: play the machine believed to have the highest payout
- Exploration: play untested machines to learn if there are higher-paying ones

Thus, the best long-term strategy may involve short-term sacrifices. The octopus will need to explore pulling the levers of several machines before it can start to exploit the best one.

4.3. Linear Contextual MAB: Introduction

Here, we used a `linear' version of the MAB algorithm, which is depicted in Figure 10. Given a context, the linear MAB uses several linear regression models (one for each action) to predict the reward for the (state, action) pair. Specifically, for each training sample comprising state, action and the corresponding reward, the linear contextual MAB trains a linear regression model that maps state to reward. Note that with the data sample comprising action index `k', only the action model `k' is trained. At test (inference) time, the bandit algorithm predicts the reward for each action model and thus can rank the actions based on the rewards yielded. Picking the best action all the time creates the exploitation scenario. Exploration is achieved by either using an upper confidence bound on the regression fit or using an ε-greedy policy, which is a way of selecting random actions with uniform distribution from a set of available actions.

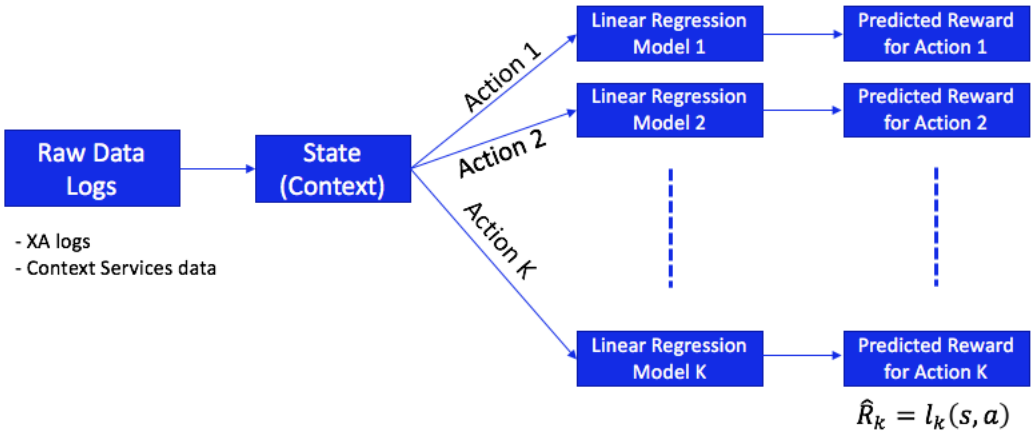


Figure 10: Linear MAB Machine Learning Flow

In general, the linear regression models may be replaced by any ML model, such as random forest, fully connected networks or generic neural networks, to create non-linear versions of the bandit architecture.

4.4. Linear Contextual MAB on the XA App: Experimental Setup

We now present the results on the linear MAB policy evaluation. This experiment was conducted using data gleaned from the Xfinity Assistant (XA) app. The problem was to correctly present the actions (or buttons) to a customer, on the XA app, that s/he would most likely click on while navigating through the app. The more relevant the presented actions are, the more engaged the customer will be, and the more likely his/her question or concern is successfully addressed.

The state, action, and reward details for this problem is described below.

State: We used the raw data from both the XA app logs and context services data. The context services data contains anonymized and individualized information such as service mix,

appointment events and present and future outage information. From this raw data, we extracted features to use as state (context) for our linear regression models.

Reward: In the XA logs, we used the following scheme to assign rewards to buttons. In the set of presented buttons,

- If the user clicked any of the buttons, that specific button gets a reward of +1.0, and other presented buttons get rewarded +0.5.
- If the user did not click on any button (which may happen if the user uttered something else or abandoned/closed the app), all buttons get rewarded -1.0.

Action: There are over 300 types of buttons in the button catalog. We only considered the 100 most frequently occurring buttons as our actions, and grouped all remaining actions into another action bucket. This means, in the context of Figure 11, we would be training 101 action models, i.e., K = 101.

4.5. Linear Contextual MAB on the XA App: Policy Evaluation Results

Given the data represented by the (state, action, tuple) pairs, we can provide results on training and policy evaluation [2]. In order to study how the bandit algorithm learns over time, we used the following methodology: In each iteration, we trained the bandit on 25,000 data samples and subsequently evaluated it on 2,500 test samples. We repeated this for roughly 1,000 iterations (which equates to one epoch for the amount of data we have). For evaluating the policy, we used the following unbiased offline evaluation methodology: When the action chosen by the algorithm matches the action chosen by the user in the data log, we add the corresponding reward. Otherwise, the data sample is ignored.

Figure 11 plots the net rewards obtained by the linear MAB over time. The increasing overall reward means that the algorithm is learning to explore and exploit the various patterns in the data over time. When we examined more closely into the positive and negative rewards that make up the net rewards (see Figure 12), we noticed that the algorithm was attempting to increase the positive rewards and avoid the negative ones, which is what we desired.

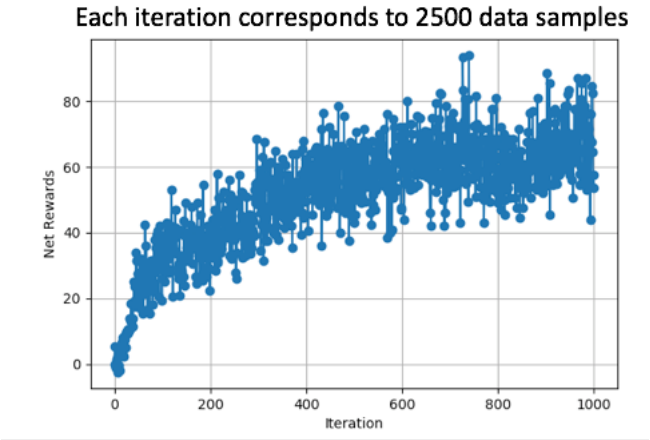


Figure 11: Linear MAB Policy Evaluation: Net Rewards

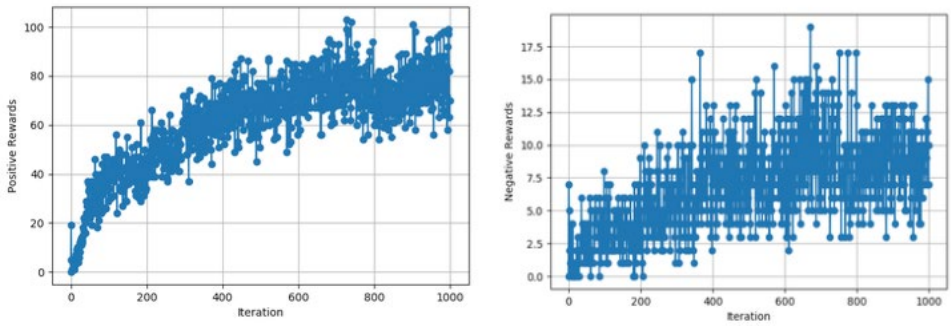


Figure 12: Linear MAB Policy Evaluation: Positive (left) and Negative (right) Rewards

Conclusion

The AI/ML building blocks described in this paper are currently being tested in a production environment, and some of the early results presented here have been obtained on live data. These results are encouraging and show that the productization of the whole Chatbot is just a matter of time.

There is actually a good fit between existing AI/ML methods and the cable telecommunication industry. We generate an overwhelming wealth of data, the interpretation of which far exceeds human capabilities. Our current AI/ML methods are barely scratching the surface of the possible, and focus on low hanging fruit to equal the performance of existing customer care. The example of the silent sufferers described in this paper is just an illustration of a case where AI/ML can reach beyond existing human resources to detect issues that are invisible to the human eye and simple query/filter systems or supervised learning. Unsupervised and reinforcement learning is well positioned to open new and radically beneficial frontiers in the operational transformation of the cable telecommunication industry.

Abbreviations

AI	artificial intelligence
DE	decision engine
HSD	high speed data
MAB	multi-armed bandit
ML	machine learning
NLP	natural language processing
NPS	net promoter score
RDK	reference design kit, http://rdkcentral.com/
XA App	Xfinity assistant app
XRE	cross-platform Runtime Environment

Bibliography & References

1. *L. Li, W. Chu, J. Langford, and R. E. Schapire, ``A Contextual-Bandit Approach to Personalized News Article Recommendation,'' In the 19th International Conference on World Wide Web (WWW), 2010.*
2. *L. Li, Wei Chu, John Langford, and Xuanhui Wang, ``Unbiased Offline Evaluation of Contextual-Bandit-based News Article Recommendation Algorithms,'' In the 4th ACM International Conference on Web Search and Data Mining (WSDM), 2011.*
3. *Binkowski, Mikolaj, Gautier Marti, and Philippe Donnat. "Autoregressive Convolutional Neural Networks for Asynchronous Time Series." 2018.*

